Please amend the specification as follows:

In the title:

~~TEMPORAL~~ CONTEXT PROGRAMMING IN OBJECT-ORIENTED ENVIRONMENTS

Page 1, lines 16-19:

This invention relates to computer programs for managing data entities or objects and, more particularly, to computer programs for managing data entities or objects that can, need, or should manage time-structured or other context data, with a seamless ability to persist to an industry standard relational database.

Page 3, line 20 to page 4, line 13:

The paper Suzuki et al., "Development and Performance Analysis of a Temporal Persistent Object Store POST/C++," Doctoral Degree Program in Engineering, University of Tsukuba, Tennohdai, Tsukuba, Ibaraki 305 Japan suzu@dblab.is.tsukuba.ac.jp (January 30, 1996) describes the use of Object Database Management Systems (ODBMS) in advanced database applications to manage complex objects and objects that have associated inherent procedures or methods. According to this paper, some applications require temporal and historical information management in ~~a~~ the context of such object management. To accomplish this, a temporal persistent object model is proposed This model treats past states of persistent objects as different objects, and supports uniform manipulation of current and past states of objects. An implementation of a temporal persistent object management system, named POST/C++, is described. This system stores objects based on C++ and manages their histories of updates by user transactions based on the temporal persistent object model. Since each store

is of a new object, the system is not very memory efficient. Further, while proposing a uniform way of dealing with (or manipulating) the temporal data, the paper does not propose a uniform way of dealing with (or manipulating) the temporal data stored while it is in use in an actual running computer program, especially one designed using an object-oriented language.

Page 4, line 14-16

It would be advantageous if there were a unified way or common programming model for use by application developers in managing context date, e.g., time or temporal data information and point of view information.

Page 4, line 19 to page 5, line 3:

The present invention is directed to control of time structured or other context related data by adding a context in object-oriented ~~object oriented~~ programming environments. In particular, the present invention provides a generic mechanism for managing "temporal" data in object-oriented environments. This makes the problem of managing such data very consistent and manageable in programs written in disparate locations, by disparate programmers, at disparate times. It allows users to easily manage this type of complexity for superior information systems – as opposed to just ignoring this element, which exists in ~~of~~ almost all business systems.

Page 6, lines 3-12:

In addition to creating a temporal database, an exemplary embodiment of the present invention also utilizes temporal methods or functions, i.e., methods or functions that may change over time or at least select data objects for processing based on a specified time and the temporal context stored with various versions of the data object. When a function call is made, according to the present invention a particular context, e.g., time, is stated as well as the object to be operated upon. The function then operates on those data objects that are valid for the specified time. Thus, if ~~it~~ three years ago, Joe was a supervisor and today he is a manager. If the function "list position" is caused to operate on the object Joe, if the time is listed as three years ago, the output will be "supervisor." If the no time is entered, indicating the default time of now, the answer to the output for the same function will be "manager."

Page 8, lines 8-19:

As a consequence, the invention broadly relates to an object-oriented context database system which includes data objects, where each data object defines a class of object with attributes. Preferably, at least one attribute of one data object is stored or used with the data object in the database along with an indication of the context (e.g., version 1) of the attribute. Any difference in the attribute's value (e.g., version 2) is also stored in the data object along with an indication of the context of the change in the attribute (e.g., a version number). At least one of the methods which the class of objects can carry out has an argument which is an indication of context (version number). A method executed with a particular context argument utilizes the attributes of the affected data objects in effect for the particular context (e.g., version) and the method is carried out according to the parameters of the method in that specified context. Either the data attributes or the method may be made to vary with context, but it is not necessary that

they both vary. If ~~In~~ nothing is specified, the current context is assumed and the most recent attributes are used.

Page 9, lines 17-20:

Figure 1 is an illustration of the structure of a conventional data object in an object-oriented ~~object oriented~~ programming environment at run time;

Figure 2 is an illustration of the structure of a data object according to the present invention in a temporal object-oriented ~~object oriented~~ programming environment at run time;

Page 15, line 17 to page 16, line 3:

Fig. 1 shows the structure of a conventional data object in an object-oriented programming ~~(OPP)~~ (OOP)environment at run time. The data object may be, for example, an employee of a company. Thus, the type or class of the object would be "employee." There are certain fields or attributes associated with the class employee. These could include name in Field 1, address in Field 2, etc. In an OOP ~~OPP~~ system the object would be stored in database. At runtime, if an operation were being performed on the object, i.e., a function call identified the object, the system would first look up the type or class of the object, which would be fixed or static. Then the fields of the class would be determined, which again would be static. The values of the fields would then be retrieved from the database.

Page 16, lines 7-16:

If the number of fields of a class or the values of those fields were to change over time, in conventional OOP ~~OPP~~, a new type object is created by overwriting the old one. Thus, the system is incapable of being reset to an earlier point in time. The Suzuki paper identified above, calls for the storing of old versions of the object when there is a change in the value of one of its fields. Then, when information about a prior time is requested, e.g., the employee's prior address, it is at least available. However, other than an index that identifies the older versions of the object, the Suzuki paper still leaves it to the programmer to deal with this temporal information. Thus, no standardized means for dealing with context in general and temporal context in particular is known in the prior art. In addition, in this paper there is no concept of absolute time. It uses a floating time "now" which is the current value of the attribute. Thus, the system cannot handle future time.

Page 17, lines 4-10:

At runtime, when using the present invention, function calls define the context, e.g., time, of interest. If no time is specified, the current system time is assumed. In response to the function call identifying a particular context, e.g., time, the system looks up the class or type of the object at that time. It then looks up the field values in effect at that time for the class, and retrieves those values. Thus, the object class as well as the values for the attributes may be different over time. However, the database stores this information in such a way that all the programmer has to do is specify a function call and the context, e.g., time, and the context (temporal) problem is handled in a standard way by the system.

Page 42, line 23 to page 43, line 5:

ADD_NEW_ATTRVAL_SHIFT - This command creates a new entry in the attribute value history of the specified attribute of the specified instance, shifting the existing attribute values according to the rules below. If the from-to range of this new attribute value lies within the from-to range of an existing attribute value (or coincides exactly), this flags an error. If the from-to range of this new attribute value subsumes one or more existing attribute values (or coincides exactly with BOTH of their endpoints - i.e. the earliest date-from and the latest date-to), this flags an error. The three illustrations of Fig. 3 show ~~shown~~ these error conditions, i.e., subsume 1 and 2, and subsumed-By.